Rapport Projet 2 d'algorithme et structure de donnée

Dumont Thomas, Chauvin Leeloo

December 2017

1 Définition des structures

Le code suivant nous a été donné dans l'énoncé.
Il permet de faire des structures pour notre chainage.

1.1 t_coord

```
struct t_coord {
    double valeur;
    int indice; // 1 a D
    t_coord * suiv; // coordonnee suivante dans le chainage
};
```

1.2 t_vecteur

```
struct t_vecteur{
    int dimension; // positive
    double defaut; // valeur par défaut
    t_coord * tete;
//chainage des coordonnees significative, ordonnees par indice croissant
};
```

1.3 Procédure initialiser

On défini comme demandé une procédure pour initialiser un vecteur.
La procédure prend comme paramètre:

- un vecteur en modifiable
- une dimension d
- une valeur par défaut du vecteur

La procédure va regarder si la dimension donnée est positive.Dans le cas contraire il nous indique qu'il y a une erreur.Si la dimension est la bonne on va donc appeller la fonction supprimer dont on parle juste après. Le principe de cette fonction est bien de supprimer tout les maillons qui pourraient exister. Plutôt d'utiliser vider qui transformerait tout les maillons pour la valeur par défaut ici on choisie de vraiment supprimer les maillon, laissant sur le pointeur de tête nullptr. Ainsi on est sur que l'initialisation sera faite sur un vecteur, qui est fraichement initialisé. On initialise la dimension avec le d donné.On initialise la valeur par défaut avec le v donné.La tête du vecteur va prendre un nouveau t_coord.On crée un pointeur vers un t_coord qui pointe vers la tête du vecteur.

Puis pour i allant de 0 à d−2 par pas de 1 on execute les actions suivantes:

- la valeur est donné par rapport à la valeur par défaut
- l'indice est donné par i+1 puisqu'il va de 1 à D et non 0 à D−1

- le maillon suivant est un nouveau t_coord
- le pointeur va pointer vers le maillon suivant

On choisie d−2 car on veut que le dernier maillon ait comme suivant nullptr. Comme on va de 0 à d−2 cela revient à aller de 1 à d−1, d étant la dimension. On fait donc en dehors de la boucle l'initialisation du dernier maillon. Avec bien entendu le nullptr.

```
void initialiser (t_vecteur & vec, int d, double v){
    if (d > 0)
    {
        supprimer(vec);
        vec.dimension = d;
        vec.defaut = v;
        vec.tete = new t_coord;
        t_coord * temp = vec.tete;
        int i;
        for ( i = 0 ; i < d-2 ; i++){
            temp->valeur = vec.defaut;
            temp->indice = i+1;
            temp->suiv = new t_coord;
            temp = temp->suiv;
        }
        temp->valeur = vec.defaut;
        temp->indice = i+1;
        temp->suiv = nullptr;
    }else{
        cout <<"erreur la dimension n'est pas strictement positive." << endl;
    }
}
```

1.3.1 Procédure supprimer

Nous avons ajouté une procédure qui a pour but de supprimer tout les maillons d'une chaine.On vérifie si le vecteur contient un maillon suivant.Si c'est le cas notre vecteur temporaire copie l'adresse de la tête et ce déplace d'un maillon.La nouvelle tête temporaire est donc sur le maillon suivant.Puis on récursive ça pour arriver sur le dernier maillon du vecteur.Arrivé sur le dernier maillon on le supprime simplement.On indique qu'il est nullptr et on remonte grâce à la récursive. Voilà le code de la procédure:

```
void supprimer (t_vecteur & vec) {
    t_vecteur temp;
    if (vec.tete->suiv != nullptr){
        temp = vec;
        temp.tete = temp.tete->suiv;
        supprimer(temp);
    }
    delete vec.tete->suiv;
    vec.tete = nullptr;
}
```

Le choix de l'implémentation de cette procédure nous permet de s'assurer qu'un vecteur ne contiendra seulement un nullptr comme pointeur de tête, et ainsi supprimer tout le vecteur. Il est utilisé uniquement dans l'initialisation et n'est normalement pas accessible à l'utilisateur.

1.4 Procédure vider

La procédure vider prend un vecteur comme paramètre.On fait un vecteur temporaire.S'il existe un maillon suivant dans le vecteur.On va donc déplacer notre maillon de manière récursive.Et pour chaque maillon, de manière récursive, on lui met la valeur par défaut du vecteur.

```
void vider (t_vecteur vec) {
    t_vecteur temp;
    if (vec.tete->suiv != nullptr){
        temp = vec;
        temp.tete = temp.tete->suiv;
        vider(temp);
    }
    vec.tete->valeur = vec.defaut;
}
```

1.5 Procédure modifier

La procédure rentre en parametre un vecteur, un indice de la positon de l'élément à modifier et la valeur de modification. On commence par les vérifications de base:

- l'indice est pas supérieur à la dimension du vecteur.
- l'indice est strictement positif.

Dans le cas contraire on retourne des messages d'erreur.On se déplace de manière récursive au maillon d'indice demandé.Et une fois sur le bon maillon, on change simplement sa valeur.
Cette procédure a un coût temporel un peu plus important avec un coût de 2+[2 ou 1+[2ou 3+[5+n appel récursif ou 3]]]. Le choix de l'appel récursif nous semblait le plus logique car dans le cas le plus court la procédure ne nous coûte que 4t.

Voilà la procédure comme suit:

```
    void modifier (t_vecteur vec, int ind, double val) {
    t_vecteur temp;
    if (vec.dimension < ind){
        cout <<"erreur l'indice renseigne
        doit etre inferieur ou egal
        a la dimension du vecteur." << endl;
    }else{
        if (ind <= 0){
            cout <<"erreur l'indice renseigne doit etre strictement positif." << endl;
        }else{
            if (ind != vec.tete->indice){
                temp=vec;
                temp.tete = temp.tete->suiv;
                modifier(temp,ind,val);
            }else{
                vec.tete->valeur = val;
            }
        }
    }
}
```

2 Entrées/Sorties

2.1 Procédure saisie

Il est donc temps de faire les saisies. On a donc fait une procédure qui prend un vecteur en entrée et qui pour chaque maillon va vérifier si sa valeur est égale à la valeurs de saisie par défaut. Si c'est le cas on va tester jusqu'à ce que notre boolean qui ne deviendra vrai que si la saisie est correct de faire les saisies de réel. La gestion de la saisie n'est pas quelque chose dont nous avons l'habitude en C++ en effet, lors du cours de POO nous avons vu à plusieurs reprises et nous avons même eu un cours travaux pratique complet sur la gestion d'exception lors des saisies. Ce n'est pas le cas pour le cours l'algorithme et nous avons donc dû trouver par nos propre moyen comment faire pour que le programme gére quand l'utilisateur donne une chaîne de caractère au lieu d'un réel.
Voilà le comme comme suit:

```
void saisir (t_vecteur vec){
    t_coord * Tempo= vec.tete;
    for (int i = 0; i < vec.dimension ; i++){
        if(Tempo->valeur == vec.defaut){
            bool valid;
            double a;
            do{
                valid = false;
                cout << "Saisir la valeur de la coordonnee d'indice: "<< i+1 << endl;
                cin >> a;
                if(cin.fail()){
                    valid = cin.fail();
                    cin.clear();
                    cin.ignore();
                    cout << "la valeur de la coordonnee doit etre un reel."<< endl;
                    }
            }while(valid);
            Tempo->valeur = a;
            Tempo = Tempo->suiv;
        }
    }
}
```

2.2 Procédure afficher

Après avoir pu faire les saisies modifications, initialisation et autre il faut donc passer à l'affichage. Pour cela nous avons donc fait une procédure prennant un vecteur et un booléan pour dire si le vecteur est complet, comme demandé dans l'énoncé. On fait un premier test si le vecteur est complet. si c'est le cas on va tout simplement afficer à l'aide d'une boucle pour basique. Sinon on va parcourir le vecteur, si la valeur d'un maillon est différent de la valeur de base on l'affiche, sinon on passe au suivant. Et ce pour toute la dimension du vecteur. Nous sommes ici sur la fonction la moins optimisé de toute car nous parcourons dans tout les cas la totalité du vecteur avec des boucles. Pour un vecteur d'un million en dimension, le programme mettrait des années pour tourner correctement. Nous n'avons pas eu le temps d'optimiser cela, mais en utilisant une boucle tant que, ou en récursif cela permettrait de faire passer la conditionnel plus rapidement et les tours de boucle aussi.

Voilà le code de la procédure d'affichage:

```
void afficher (t_vecteur vec, bool complet){
    if(vec.tete!=nullptr){
        t_coord * temp = vec.tete;
        if(complet){
            cout<<"[";
            for (int i = 0; i < vec.dimension ; i++){
                cout<<temp->valeur;
                temp = temp->suiv;
                if(i < vec.dimension-1){
                    cout<<" , ";
                }
            }
            cout<<"]"<<endl;
        }else{
            bool virg = false;
            cout<<"[";
            for (int i = 0; i < vec.dimension ; i++){
                if(temp->valeur != vec.defaut){
                    if(virg){
                        cout<<" , ";
                        virg = false;
                    }
                    cout<<temp->indice<<":"<<temp->valeur;
                    virg = true;
                }
                temp = temp->suiv;
            }
            cout<<"]"<<endl;
        }
    }else{
        cout<<"La tête du vecteur est vide"<<endl;
    }
}
```

3 Opération algébriques

3.1 somme

Il est donc maintenant de faire des opérations sur les vecteurs. La première une somme. Rien de plus facile, on prend en entrée deux vecteurs. S'ils sont de même dimension et d'une dimension supérieur à zéro, on va pouvoir faire la somme, sinon on affiche un message d'erreur. On va donc initialiser le nouveau vecteur qu'on va renvoyé, dans le cas où l'on peut faire la somme. Sa valeur par défaut est donc la somme des valeurs par défaut. Puis on va se balader tout du long des vecteurs en ajoutant à notre nouveau vecteur des t_coord qui vont prendre la somme des deux premiers vecteurs. On va cependant pour gagner en optimisation faire le test de si la valeur par défaut est différente de la valeur, pour les deux vecteurs. Si ce n'est pas le cas on suppose que l'un des deux vecteurs n'a pas de maillon suivant qui contient une valeur, quelque soit le maillon.
Le coup temporel est de [26+[30+[n appel récursif de la fonction] ou 0] ou 9]+1 t.

```
t_vecteur somme (t_vecteur a, t_vecteur b){
    t_vecteur c;
    if(a.dimension == b.dimension && a.dimension > 0){
        c.tete = new t_coord();
        c.dimension = a.dimension;
        c.defaut = a.defaut + b.defaut;
        if(a.defaut!=a.tete->valeur  && b.defaut != b.tete->valeur){
            c.tete->indice = a.tete->indice;
            c.tete->valeur = a.tete->valeur + b.tete->valeur;
            c.tete->suiv = new t_coord();
            a.tete = a.tete->suiv;
            b.tete = b.tete->suiv;
            c.tete->suiv = (somme(a,b)).tete
        }else{
            cout<<"Les vecteurs ne contiennent plus de valeurs"<<endl;
            c.tete = nullptr;
            c.dimension = 0;
            c.defaut = 0;
        }
    }else{
        cout<<"les vecteurs doivent etre de meme dimension"
        +"et non nulle pour que leur somme ait un sens"<<endl;
        c.tete = nullptr;
        c.dimension = 0;
        c.defaut = 0;
    }
    return c;
}
```

3.2 produit

La fonction suivante nous permet de retourner le produit scalaire de deux vecteurs. Pour cela on test dans un premier temps si les dimensions des vecteurs sont les mêmes et si elles sont bien supérieur à zéro. Si c'est le cas alors nous pouvons donc vérifier si la valeur est différente de la valeur par défaut, si c'est le cas encore une fois on applique l'opération on passe au maillon suivant. Et on fait par récurence la somme de p et s'il existe la multiplication de la valeur suivante de a et b. Si ce n'est pas le cas, nous supposons qu'il n'y a pas de valeur suivante dans le vecteur et nous arretons donc le calcule. Et enfin si les dimensions ne sont pas positive ou ne sont pas égale alors nous retournons un message d'erreur.
Le coup temporel est de [31+[n appel récursif de la fonction] ou 5]+1 t.

```
double produit (t_vecteur a, t_vecteur b){
    double p;
    if(a.dimension == b.dimension && a.dimension > 0)
    {
        if(a.defaut != a.tete->valeur && b.defaut != b.tete->valeur)
        {
            p = a.tete->valeur * b.tete->valeur;
            a.tete = a.tete->suiv;
            b.tete = b.tete->suiv;
            p = p + produit(a,b);
        }else{
            cout<<"Les vecteurs ne contiennent plus de valeurs"<<endl;
            c.tete = nullptr;
```

```
            c.dimension = 0;
            c.defaut = 0;
        }
    }else
    {
        cout << "les vecteurs doivent etre de meme dimension"
        +"et non nulle pour que leur produit scalaire ait un sens."
        +"Attention la fonction retourne 0 par defaut !" << endl;
        p = 0;
    }
    return p;
}
```

4 Main

Voilà le main que nous avons utilisé pour faire tourner le programme. Nous avons rencontrer des problèmes lors de la compilation. En effet dans certains cas et sous certaines configuration si le cout<<""; n'était pas présent le programme s'arrêtait avec des messages d'erreurs. Nous avons chercher pour trouver d'où cela pouvait venir, mais sans succès. Nous avons aussi eu quelques erreurs de segmentation. Probablement dû a la tentative d'accès avec un pointeur vers de la mémoire non alloué. Mais normalement dans la version que nous vous avons fournit ce genre d'erreur de doit pas arriver. Un autre problème que nous avons rencontré, qui n'était pas mentionné et que nous n'avons pas su comment le gérer: le cas de la valeur par défaut. Un utilisateur rentre 2 comme valeur par défaut, cela implique qu'il ne pourra jamais utiliser de 2 dans son vecteur. Ce qui dans certains cas peut sembler fort peu logique. Nous avions aussi penser à faire une procédure, que nous n'avons malheureusement pas eu le temps de faire pour faire des copies de vecteur.
Nous avons essayé d'utiliser un maximum les fonctions récursives et les boucles tant que, pour optimiser le code un maximum et ne pas avoir à parcourir le vecteur un milliard de fois si la dimension du vecteur est d'un milliard.

```
int main(){
    t_vecteur a;
    a.tete = nullptr;
    initialiser(a,3,2);
    t_vecteur b;
    b.tete = new t_coord;
    initialiser(b,3,2);
    cout<<"";
    afficher(somme(a,b),true);
    return 0;
}
```

Comme promit le programme si dessus nous donnes un de nos messages d'erreur en effet, on lui donne un vecteur qui est la somme de deux vecteur non modifier. Sa tête est donne nullptr et la fonction afficher devrait nous dire qu'elle ne peut pas le faire.